A residual inspired approach for fault localization in DES

Matthias Roth, Jean-Jacques Lesage, Lothar Litz

A residual inspired approach for fault localization in DES

Matthias Roth*,. Jean-Jacques Lesage**,**
Lothar Litz*

**Institute of Automatic Control, University of Kaiserslautern,*
P.O. Box 3049, 67653 Kaiserslautern, Germany, {mroth, litz}@eit.uni-kl.de
***LURPA – Ecole Normal Superieur de Cachan*
61, Avenue du Président Wilson, 94235 Cachan Cedex, France, {roth, lesage}@lurpa.ens-cachan.fr

Abstract: In this paper an approach for fault localization in Discrete Event Systems (DES) is proposed. The presented diagnosis method allows fault localization using a fault-free nominal system model. Via a systematic comparison of the observed and the expected system behavior, it is possible to determine a set of fault candidates. Inspired by residuals known from diagnosis in continuous systems, different set operations are presented that carry out this comparison. After a fault has been detected and a first estimate concerning its localization has been performed, a special algorithm analyzes the further system behavior in order to determine a more precise fault localization. The algorithm also works on the basis of the nominal system model. The method is explained using a manufacturing system example.

Keywords: Discrete Event Systems, Fault Detection, Fault Diagnosis

1. INTRODUCTION

Diagnosis in discrete event systems (DES) has been a vital research area for the last 15 years. One of the main purposes of diagnosis methods is to increase the availability of technical systems. An increased availability has positive effects on system dependability and economic key issues such as productivity.

An important class of DES diagnosis approaches is model based. The idea behind is to compare the modeled and the observed system behavior in order to detect and localize the faults. Model-based approaches can be divided in two groups. The first group considers models that contain fault-free behavior as well as system behavior for given faults. The second possibility is to use models of fault-free system behavior only.

A prominent example for approaches with models including the faulty behavior is given in [Sampath, et al., 1996]. One of the main features of this "diagnoser"-approach is the possibility to give guarantees concerning the diagnosabilty of faults that are considered in the underlying model. If certain conditions hold, faults that are considered in the model can be precisely localized. A disadvantage of this class of approaches is that only faults that actually have been considered in the system model can be detected and localized. Diagnosis methods working with a model that only contains the nominal, fault-free system behavior avoid this disadvantage. All faults that lead to a deviation from the nominal behavior can be detected when comparing modeled and observed behavior [Cordier, et al., 2004]. The drawback of this second class of approaches is due to the fact that the models have been built using less knowledge (only the nominal system behavior): Fault localization is more difficult and not always possible. In this paper, a method is proposed that yields fault localization for diagnosis systems working with a fault-free system model.

The paper is structured as follows: In section 2 an example to illustrate the proposed method is introduced. It is very similar to a case study of [Philippot, et al., 2007] where a timed decentralized adaptation of the diagnoser approach is proposed that also deals with diagnosis in manufacturing systems. In section 3 formal definitions concerning the models of the nominal system behavior are given. Section 4 deals with the proposed "residual-inspired" fault localization method. In order to deliver more precise information about fault localization, section 5 introduces a special observer algorithm that is carried out after fault detection.

2. PRESENTATION OF AN EXAMPLE

The proposed method will be explained using an example of a manufacturing system. Its purpose is to sort parcels according to their size. The programmable logic controller (PLC) that controls the system has 9 inputs and 4 outputs. Fig. 1 shows the system: a parcel is transported to the sorting station using conveyor 1. Based on the optical sensors k_1 and k_2 it can be decided if the parcel is a large one (k_1=1 and k_2=1) or if it is a small one (k_1=1 and k_2=0). Large parcels are pushed to conveyor 3, small ones to conveyor 2 using the double-acting cylinder A. Setting the output A+ makes the cylinder extend (A+=1, A-=0), resetting A+ and setting A- makes the cylinder retract (A+=0, A-=1). When the parcel arrives at the appropriate position, it is pushed by one of the single-acting cylinders B or C on the according conveyor. The cylinders B and C can be extended by setting the outputs B (B=1) or C (C=1). If they are reset, the cylinders move back to their initial position. Throughout the paper, the value "1" represents an actuator that has been set or a sensor that is

activated. “0” represents a reset actuator and a deactivated sensor respectively.

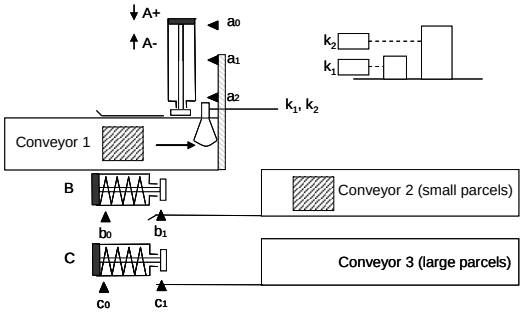

Fig. 1: Case Study: Parcel sorting station

It is assumed that the considered system is programmed such that several components work concurrently: while cylinder A is retracted, one of the cylinders B or C can be extended.

Classically, diagnosis is carried out in two steps: the detection of a fault and in case of detection, fault localization. We are now going to explain the nominal system model that is used in our approach and how it is obtained.

3. NOMINAL DES MODEL

3.1 Data-collection

The diagnosis of a DES like in Fig. 1 using a model-based approach necessitates sending the I/O vectors (input/output) of the PLC to a diagnosis system. Fig. 2 shows the principle of such a data collection. When the input information of the sensors is read, the setting of the controller outputs is determined by executing the control program. At the end of the program execution the current I/O vector containing both the last input information and the newly determined output information is sent to the diagnosis system.

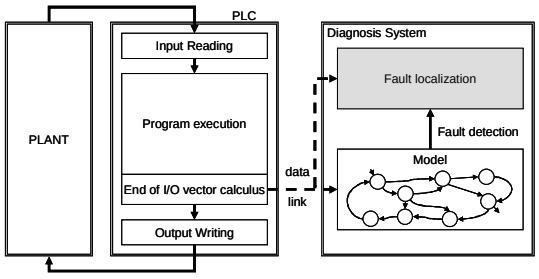

Fig. 2: Data-collection for diagnosis

The system to be diagnosed is a closed loop consisting of PLC and plant that can be observed by the evolution of the I/O vectors at the end of each program execution of the controller cycle (Fig. 2). The model that will be used for diagnosis is based on these I/O vectors. It is built using an identification approach given in [Klein, et al., 2005].

Definition 1 (I/O vector definition): The *j*-th I/O vector is defined as $u(j) = (I_1(j), ..., I_s(j), O_1(j), ..., O_m(j))$ with I_1, .., I_s and O_1, .., O_m denoting the considered inputs and outputs of the closed loop system. The inputs and outputs can have the values “1” or “0”. $|u(j)|$ denotes the length of the vector (number of I/Os).

3.2 Model definition

The identification algorithm that works on the basis of the I/O vectors delivers an automaton of the Moore type. It has an output alphabet which contains the possible I/O vectors of the system. Due to the non-deterministic nature of the considered class of systems, the automaton is denoted a **N**on-**D**eterministic **A**utonomous **A**utomaton with **O**utput:

Definition 2 (NDAAO): NDAAO = $(\mathbf{X}, \mathbf{\Omega}, r, \lambda, x_0)$ with

$\mathbf{X}$	finite set of states,
$\mathbf{\Omega}$	output alphabet,
$r : \mathbf{X} \rightarrow 2^{\mathrm{X}}$	non-deterministic transition relation,
$\lambda : \mathbf{X} \rightarrow \mathbf{\Omega}$	output function,
x_0	initial state.

During diagnosis, the dynamics is defined as follows: given a current state $x(i)$ with output $\lambda(x(i)) = u(i)$ and a new observed I/O vector $u(j)$ different from $u(i)$. The automaton tries to find a state $x(j)$ such that $x(j) \in r(x(i))$ with $\lambda(x(j)) = u(j)$. If several states fulfill this condition, the choice is not deterministic. If none of the successor states has the observed output $u(j)$, the automaton can not proceed in a valid state and a fault is detected.

The second step of diagnosis is to determine which I/O(s) is (are) possibly responsible for a detected fault. Thus it is necessary to analyze which I/Os change their value from one vector to another. For this purpose a function is introduced that delivers the rising and falling edges of the changing I/Os when two I/O vectors are compared. In the following $u(j)[i]$ denotes the I/O at the *i*-th position of the *j*-th I/O vector, i.e. the I/O with the index *i*.

Definition 3 (I/O label): IO_i is defined as the label or name of the I/O at the *i*-th position of an I/O vector *u*(*j*).

Definition 4 (Edge function): Let $u(j), u(k)$ be two I/O vectors of the considered system. The function

$$Edge(u(j)[i], u(k)[i]) = \begin{cases} IO_i\_1 \text{ if } u(j)[i]=0 \ \wedge \ u(k)[i]=1 \\ IO_i\_0 \text{ if } u(j)[i]=1 \ \wedge \ u(k)[i]=0 \\ IO_i\_\varepsilon \text{ if } u(j)[i]=u(k)[i] \end{cases}$$

delivers the rising and falling edges of *i*-th I/O resulting from the comparison of $u(j)[i], u(k)[i]$. The rising and falling edges are named according to the label of the related I/O. If the analyzed I/O does not change its value, the symbol IO_i_ε is delivered.

Fig. 3 shows an example of the *Edge* function. The considered I/O vector has I/Os with labels *a*, *b* and *c* and with the indices 1, 2 and 3. The consecutive I/O vectors $u(1)$ and $u(2)$ are compared to get the rising and falling edges. Note that two consecutive I/O vectors can lead to the observation

of several rising and falling edges if the according I/Os changed their value within the same cycle of the PLC. Let *a* be the label of an input with a falling edge that causes the control program to set the output *b*. Due to the data collection process described in section 3.1, the two edges *a_0* and *b_1* will be observed simultaneously between the two I/O vectors.

$$u(k) = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$u(1) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \text{ and } u(2) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \Rightarrow \begin{cases} Edge(u(1)[1], u(2)[1]) = a\_0 \\ Edge(u(1)[2], u(2)[2]) = b\_1 \\ Edge(u(1)[3], u(2)[3]) = c\_\varepsilon \end{cases}$$

Fig. 3: Example of the Edge-function

Since it is possible to observe more than one rising or falling edge when comparing two I/O vectors, it is necessary to define an evolution set containing the edges that have occurred.

Definition 5 (evolution set):
$ES(u(j),u(k)) = \{Edge(u(j)[i],u(k)[i]) | Edge(u(j)[i],u(k)[i]) \neq IO_i\_\varepsilon \ \forall\ 1 \leq i \leq |u(j)|\}$ determines the set of rising and falling edges between two I/O vectors $u(j)$ and $u(k)$.

The *ES*-function applies the *Edge*-function to each I/O of two I/O vectors and merges the results in one set. Results of the *Edge*-function are only considered if the function does not deliver IO_i_ε, i.e. the according I/O actually changes its value. For the example in Fig. 3 the function determines $ES(u(1),u(2)) = \{a\_0, b\_1\}$.

If a system model in form of an NDAAO is used for fault localization, it is possible to apply the *ES*-function to the output of states (e.g. $ES(\lambda(x_1), \lambda(x_2))$). Thus it is possible to determine the rising and falling edges between two states.

Fig. 4 shows a part of the nominal model of the case study from Fig. 1. The model has been identified using the identification algorithm from [Klein, et al., 2005]. It has overall 38 states and 44 transitions. Due to space limitations, only the I/O vector that is the state output of x_0 is shown in the figure ($\lambda(x_0)$). The transitions in the diagram are labeled with the rising and falling edges that can be observed when applying the *ES*-function to the outputs of connected states. This allows the reconstruction of each state output. From state 4 to state 5 e.g. a falling edge of output A+ (extend cylinder A), and rising edges of A- (retract cylinder A), B (extend cylinder B) and a1 (sensor at the middle position of cylinder A) have been observed ($ES(\lambda(x_4), \lambda(x_5))$ ={A+_0, A-_1, B_1, a1_1}).

Fig. 4 shows that different sequences are possible during a fault-free system evolution due to the temporal non-determinism of the plant behavior. From state 5 it is e.g. possible to have a falling edge of *a1* before a falling edge of *b0* or vice versa: if cylinder A is retracted and cylinder B is extended, it is possible that in one system cycle cylinder A moves faster and in another system cycle cylinder B.

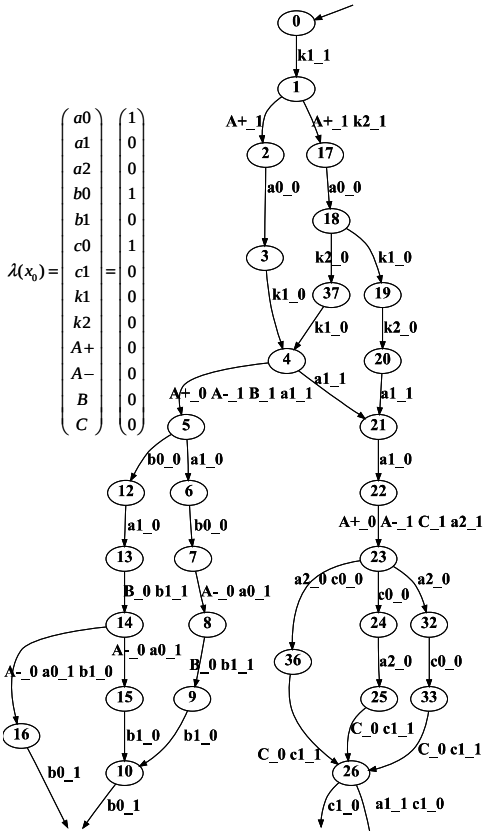

Fig. 4: Part of an identified system model

4. FAULT LOCALIZATION

4.1 Fault localization in nominal models

The system is diagnosed using the identified nominal model given by an NDAAO. The hypothesis of this approach is: "each behavior that is not reproducible by the model is a fault symptom." Hence, as soon as an I/O vector is observed that cannot be reproduced by the actual state or by one of its direct successors a fault is detected and fault localization starts.

Considering DES like the example given in section 2, fault localization techniques have the precise aim to report I/Os that are possibly related to the fault. For example in case of a fault that causes the sensor at the initial position of cylinder A to be stuck at "1", an appropriate fault localization would be "fault at sensor a1 (expected value 0, observed value 1)."

In continuous systems such localization is often carried out by determining residual signal values. (Isermann et al., 1997) give the definition of a residual as follows: "A residual is a fault indicator, based on a deviation between measurements and model-based computation."

Since the method proposed in this paper yields an analysis of the behavioral deviation between the real system and the model, several set operations will now be defined that are inspired by the definition of (Isermann et al., 1997).

In the following, u_{acutal} denotes the actual I/O vector that has led to fault detection and x denotes the actual state when the fault is detected.

4.2 Characterization of observed but unexpected behaviors

The first class of residuals that is introduced has the aim to localize faults that led to an observed behavior that was *unexpected* in the given context. The actual context is defined by the actual state x in the automaton. The first residual is defined as:

$$\text{Res1} = ES(\lambda(x), u_{actual}) \setminus \bigcup_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$$

With $ES(\lambda(x), u_{actual})$ the rising and falling edges are determined that are observed when comparing the I/O vector of the last actual state and the I/O vector that led to fault detection. This set represents what actually happened when the fault was detected. $\bigcup_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$ represents the union of the sets of rising and falling edges when the actual state and each of its direct successor states are considered. It represents the expected behavior. The set difference of the observed $(ES(\lambda(x), u_{actual}))$ and the expected $(\bigcup_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x')))$ behavior is built in the residual equation. In residual 1, the expected behavior is given by the union of each possible following behavior of the actual state. A more strict formulation of the expected behavior is used in the second residual:

$$\text{Res2} = ES(\lambda(x), u_{actual}) \setminus \bigcap_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$$

Instead of a union over the expected behavior of the possible following states, an intersection is used. The intersection delivers the edges that *must* be observed no matter which following state in the model is taken. It is obvious that $\text{Res1} \subseteq \text{Res2}$ since $\bigcup_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x')) \supseteq \bigcap_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$. As an example let us consider a fault at sensor c0. It is assumed that due to a cable break the value switches from 1 to 0 when the automaton is in state 14. This situation is depicted in Fig. 5 where a part of the nominal system model from Fig. 4 is given. A fault is detected when an I/O vector that leads to the edge {c_0} is observed from state 14.

$\bigcup_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$ = {A-_0, a0_1, b1_0} and $\bigcap_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$ = {A-_0, a0_1}. In any case, $\bigcap_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$ = {A-_0, a0_1} must be observed when state 14 is left to state 15 or 16.

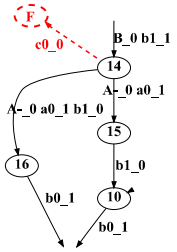

Fig. 5: Example for an unexpected behavior

For the given example, both Res1 and Res2 result in {c0_0}. The input that changed its value *unexpectedly* due to a fault is part of the residual and can be considered as possible fault localization. If the observed behavior was completely unexpected like in the example, the two residuals usually have the same result. In case that the observation partly contains expected behavior, it is possible that Res2 contains more fault candidates than Res1. In this situation it is a reasonable strategy to first check the functioning of the I/Os in Res1. If the faulty one is not contained, the supplementary I/Os of Res2 can be analyzed.

4.3 Characterization of expected but unobserved behaviors

In contrast to an observed but unexpected behavior it is also possible that a fault can be localized by determining a *missed* rising or falling edge. Set operations that help to localize an *expected but unobserved* behavior are given by the third and the fourth residual.

$$\text{Res3} = \bigcap_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x')) \setminus ES(\lambda(x), u_{actual})$$

Res3 is the set difference of the edges that are expected no matter which following state is taken $(\bigcap_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x')))$ and the edges that have been observed $(ES(\lambda(x), u_{actual}))$. Each rising or falling edge that *must* occur when the actual state is left but has not been observed is part of Res3. The expected behavior is represented by the intersection of each possible following behavior. It is also possible to give a less strict formulation of the expected behavior by using the union operation instead of the intersection:

$$\text{Res4} = \bigcup_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x')) \setminus ES(\lambda(x), u_{actual})$$

Since $\text{Res3} \subseteq \text{Res4}$, the result of Res4 is usually less restrictive than Res3, it contains more elements. The example depicted in Fig. 6 is considered to illustrate Res3 and Res4. A possible fault that can lead to the observed edges in the context of the example is sensor a1 stuck to 0. Cylinder A passes the position of a1 without the sensor changing its value. When the cylinder reaches the position of a2, the fault is detected. The residuals in this case are Res3 = {a1_1, B_1} and Res4 = {a1_1}. Input a1 is part of the two residuals and can be given as possible fault localization. It can also be seen that the edge B_1 is not always observed when state 4 is left: only if the parcel is to be pushed on conveyor 2, the output B will be set.

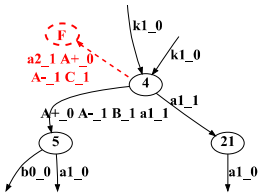

Fig. 6: Example for a missed behavior

Especially if production systems with many I/Os are considered, the residuals can help to get a relatively small set of I/Os that could be related to the fault. A maintenance operator can then check the possibly faulty sensors or the related actuators. Actuators are related to a sensor if their activation or deactivation has an influence on the sensor state.

Usually it is not known if a detected fault is to be localized by an analysis of the observed but unexpected or by the expected but unobserved behavior. Hence, each residual has to be used. If e.g. the example in Fig. 6 is analyzed using Res1, the following candidate set is determined: Res1 = {a2_1, C_1}. Together with the results of Res3 and Res4, it can not be decided if the input a2 changed its value prematurely or if the input a1 didn't change its value. Both scenarios are possible and thus, each I/O in the residuals has to be analyzed by the operator. To reduce the set of possible fault candidates, a special reduction strategy is introduced in the next section.

5. REDUCTION OF THE RESIDUAL SETS

As pointed out in the former section, applying the residuals when a fault has been detected can lead to an ambiguous situation with several fault candidates. The candidates that are part of the four residuals can be given in the following compact form:

$$\text{Candidates} = ES(\lambda(x), u_{actual}) \cup \bigcup_{\forall x' \in r(x)} ES(\lambda(x), \lambda(x'))$$

It is obvious that $\text{Candidates} \supseteq \text{Res1} \cup \text{Res2} \cup \text{Res3} \cup \text{Res4}$ since the set contains the union of the observed and the expected edges. In some cases it may be necessary that each I/O that is part of the Candidate-set (and thus the residuals) is checked by the maintenance operator. In order to give a more precise estimation of which I/O is possibly affected, it is possible to take further I/O vectors into account that follow the vector that led to fault detection.

When a deviation between modeled and observed behavior has been detected, the automaton does no longer have an actual state that corresponds to the measured I/O vector. Hence it is necessary to perform a state estimation on the basis of the I/O vectors that follow. In this paper only faults are considered that lead to a persistent change in value of an I/O. A faulty change in value of an I/O can lead to I/O vectors that do not have a corresponding state in the fault free nominal system model. In order to perform a state estimation in spite of a permanently affected I/O, it is necessary to mask out I/Os that are possibly related to the fault. Hence, each I/O with a rising or falling edge that is part of the Candidate-set should not be considered when the state estimation is performed.

Based on the outputs of the estimated states it is analyzed which of the possibly affected I/Os show a normal behavior when further I/O vectors are observed. The aim is to reduce the set of possibly affected I/Os and thus to improve fault localization. Before the candidate reduction algorithm is introduced, several definitions are necessary.

Definition 6 (IndexList): *IndexList* denotes a list containing the indices of the I/Os that are in one of the residuals 1-4 (i.e. in the Candidate-set). The index of an I/O is its position in the I/O vector (see $\lambda(x_0)$ in Fig. 4 for the considered example).

Since the possibly affected I/Os will be masked out during the state estimation, an I/O vector projection is defined.

Definition 7 (I/O vector projection): The I/O vector projection of an I/O vector *u* to a list with indices that have to be masked out (*IndexList*) is defined as:

$$IOP_{IndexList}(u)[i] = \begin{cases} u[i] \text{ if } i \notin IndexList \\ * \text{ if } i \in IndexList \end{cases} \forall i = 1,..,|u|$$

$$\text{and } IOP_{IndexList}(u) = \begin{cases} IOP_{IndexList}(u)[1] \\ ... \\ IOP_{IndexList}(u)[|u|] \end{cases}$$

$IOP_{IndexList}(u)$ contains for each I/O that has to be masked out (i.e. an I/O with an index in the *IndexList*) the don't care symbol *. For each I/O that has to be considered, $IOP_{IndexList}(u)$ has the according value of the vector *u*.

The estimation algorithm in Fig. 7 starts with analyzing the I/O vector *u(t)* which follows *u(t-1)* that led to fault detection. Firstly, the state estimation is initialized. Each state of the NDAAO is a possible estimate ($X_{t-1} := X$). In step1 the set X_t is determined. A state $x' \in X$ is added if the following conditions hold: Firstly, *x´* is a successor of one of the states from the set X_{t-1} containing the former estimation. Secondly, exactly the edges like observed between the I/O vectors *u(t-1)* and *u(t)* can be observed when the state *x´* is reached. In the next line of the algorithm, each state that has an output that differs from the observed vector *u(t)* in one of the "healthy" I/Os (that consequently is *not* part of the *IndexList*) gets removed. It is assumed that such a state cannot represent a system state that led to the new vector *u(t)* since it differs in one of the not affected I/Os. After this, the old state estimation X_{t-1} is replaced by the new one $X_{t-1} := X_t$.

In Step2 it is analyzed which I/O can be taken from the fault candidates because it showed a normal behavior. It is assumed that a possibly affected I/O worked properly if the following condition holds: the considered I/O has in each state output of the estimation the same value as in the last observed I/O vector *u(t)*. The assumption is based on the fact that the state estimation has been performed *without* considering possibly affected I/Os (with index in the *IndexList*). If a possibly affected I/O has nevertheless the same value as the observed one in each state of the

estimation, it is highly probable that it is not a fault candidate. If the condition holds, the index of the considered I/O can be taken from the *IndexList* and the I/O can also be removed from the residuals.

In Step3 it is determined how many candidates are left in the *IndexList* and if the state estimation is unambiguous. If the state estimation is unambiguous ($|X_t|=1$) and if there is no index of an I/O left in the *IndexList*, the usual system monitoring can go on using the determined state as the current state. Probably, there was a false alert. False alerts can arise if the fault-free (possibly identified) system model does not capture the whole fault-free system behavior. If there is more than one possible state in the estimation and more than one I/O left, the state estimation continues with step1 by considering the next I/O vector. If none of the two conditions can be fulfilled, the algorithm cannot reduce the candidate set by considering following I/O vectors and stops.

Algorithm 1: State estimation and candidate set reduction

Initialization: $X_{t-1} := X$

Step1: Consider the next I/O vector denoted as $u(t)$

$$X_t := \{x' \in X \mid \exists x \in X_{t-1} : x' \in r(x) \wedge ES(u(t-1), u(t)) = ES(\lambda(x), \lambda(x'))\}$$

$$X_t := X_t \setminus \{x \in X_t \mid IOP_{IndexList}(\lambda(x)) \neq IOP_{IndexList}(u(t))\}$$

$$X_{t-1} := X_t$$

Step2: For each *IOIndex* in *IndexList*:

if $\forall x \in X_t : \lambda(x)[IOIndex] = u(t)[IOIndex]$ holds

then $IndexList := IndexList \setminus IOIndex$

Step3:

if $|X_t| = 1$ and if $|IndexList| = 0$ then false alert

if $|X_t| > 1$ and if $|IndexList| > 1$ then loop back to step1

Fig. 7: State estimation and candidate set reduction

As illustration of the algorithm in Fig. 7 the example of Fig. 8 is considered. It is assumed that sensor k1 is stuck to 1 after it noticed the presence of a parcel. A fault is detected when the parcel arrives at conveyor 2 without k1 having changed its value to 0.

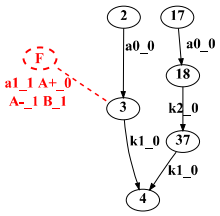

Fig. 8: Example for the candidate set reduction algorithm

Based on the residuals it is possible to get the information that one of the sensors k1 or a1 or one of the related actuators is affected, since starting from state 3 Res1 = Res2 = {a1_1, A+_0, A-_1, B_1} and Res3 = Res4 = {k1_0}.

It cannot be decided if k1_0 was missed or if a1_1 changed its value prematurely. Hence the algorithm in Fig. 7 starts. In the example it is assumed that after fault detection two I/O vectors are observed that lead to the edges {b0_0} (first vector) and {a1_0} (second vector). Fig. 9 shows the evolution of the state estimation and the reduction of the candidate set that is delivered by the algorithm (only inputs are considered in the example). The I/O vector after fault detection could possibly be represented by the two states 7 or 12 of the model in Fig. 4 since their I/O vector projection equals the projection of the measured I/O vector. Neither a1 nor k1 did yet show a normal behavior. The next I/O vector in the example leads to an unambiguous state estimation since only state 13 can be reached from one of the states 7 or 12 by producing the observed edge a1_0. In state 13 the input a1 is 0 as well as in the last observed I/O vector. Hence it can be removed from the candidate set. After the analysis of two I/O vectors only the index of the affected I/O k1 is in the *IndexList*. The algorithm led to a more precise fault localization.

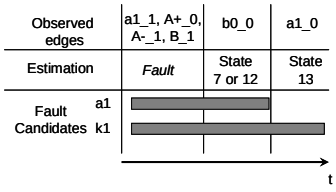

Fig. 9: Evolution of the candidate set

6. OUTLOOK

In this paper a fault localization technique that works on the basis of a fault-free nominal system model has been presented. The presented approach was developed for monolithic system models. In large applications automata networks are often used for diagnosis purposes. In future works the approach will be extended for the application in such automata networks. Furthermore, the adaptation of the approach to timed automata is part of current work.

REFERENCES

Cordier, M.-O., Dague, P., Lévy, F., Montmain, J., Staroswiecki, M., Travé-Massuyès, L. (2004). Conflicts versus analytical redundancy relations: A comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives, *IEEE Transactions of Systems, Man, and Cybernetics-Part B*, vol. 34, No. 5, pp. 2163-2177

Isermann, R., Ballé, P. (1997). Trends in the application of model-based fault detection and diagnosis of technical processes, *Control Engineering Practice 5*, pp. 709-719.

Klein, S., Lesage, J.-J., Litz, L. (2005). Fault Detection of Discrete Event Systems Using an Identification Approach. *16th IFAC World Congress*, Prague (Czech Republic) CDRom paper n°02643

Philippot, A., Sayed-Mouchaweh M., Carré-Ménétrier V. (2007). Unconditional Decentralized Structure for the Fault Diagnosis of Discrete Event Systems. *Proceedings of the 1st IFAC Workshop on Dependable Control of Discrete Systems*, Cachan (France), pp. 255-260

Sampath, M., R. Sengutpa, S. Lafortune, K. Sinnamohideen and D. Teneketzis (1996). Failure Diagnosis using Discrete-Event Models. *IEEE Transactions on Control Systems Technology*, vol 4, No. 2, pp. 105-124.